Fast Approximation of Maximum Flow using Electrical Flows

Cameron Musco

Advisor: Dan Spielman

April 18, 2011

Abstract

We look at a variety of topics relating to better understanding the fast approximate maximum flow algorithm presented in 'Electrical Flow, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs' (Christiano, Kelner, Madry, and Spielman 2010). The algorithm constructs an approximate maximum flow from a series of intermediate electrical flows over an iteratively updated resistor network. We explore the relationship between electrical flows and maximum flow and discuss a number of topics relevant to the goal of proving that the intermediate electrical flows calculated in the algorithm directly converge to a maximum flow (rather than converge as an averaged set). We present a proof that the algorithm can be run without the binary search technique described in the original paper, and discuss how the slightly modified algorithm may help better understand the overall behavior of the technique. Finally, we look at some interesting results that arise when we consider continuous time analogs of the discrete iterative algorithm. We discuss how the maximum flow on a graph can arise as a steady state flow of a related thermistor network and how this phenomena could possibly be used to design future maximum flow algorithms.

1 Background

1.1 Maximum Flow Problem

The maximum flow problem is a well studied and widely applicable problem in algorithmic graph theory and optimization. The problem is: given a graph and an associated set of flow constraints on the edges of the graph, to find the maximum amount of flow that may travel between a source node s and a sink node t. Solving this problem has applications ranging from image segmentation, to airline flight scheduling, to solving other fundamental graph based problems like the related maximal bipartite matching and perfect matching problems [12].

There are many known algorithms for solving maximum flow on a graph. The Ford-Fulkerson algorithm is a staple of undergraduate algorithms classes and two techniques - the push relabel method and the binary blocking algorithm, have been the basis for much recent work on improving runtime bounds for the problem. However, finding faster algorithms (and faster approximation algorithms) has been a long standing open question.

1.2 Overview of Technique

In 2010, Christiano et al. published a paper describing a new algorithm for computing approximate maximum flow on an undirected graph. This algorithm can compute a $(1-\epsilon)$-approximate maximum flow in time $\tilde{O}(mn^{1/3}\epsilon^{-11/3})$ - where n is the number of nodes in the graph and m is the number of edges. This algorithm currently has the best runtime dependence on m and n of any known approximate maximum flow algorithm. [2]

The algorithm works by computing a series of electrical flows on the graph in consideration. The electrical flow of value F is the flow f of F units from s to t that minimizes the energy function $\mathcal{E}_r(f) = \sum_e r_e f^2(e)$, where r_e is a resistance value assigned to each edge. This electrical flow can be solved as a system of linear equations of the graph Laplacian. Recent work by Spielman, Teng, and others has shown that, since the Laplacian is symmetric and diagonally dominant, these linear equations can be solved in nearly linear time.

Intuitively, what the algorithm does is chooses an initial set of resistances to apply to the graph. The algorithm then computes the electrical flow over the graph given these resistances. The electrical flow will not necessarily preserve the capacity constraints that must be preserved in a maximum flow. So, the resistances on edges where capacity constraints are violated are augmented in order to decrease flow across these edges. The authors show that by appropriately modifying the resistances at each iteration and by taking an average over a set of intermediate electrical flows, an approximate maximum flow is achieved.

Although a full description of the algorithm can be found in [2], I will give a brief description, without proof of correctness here.

1.3 Description of the Algorithm

Before presenting the $\tilde{O}(mn^{1/3}\epsilon^{-11/3})$ time algorithm, the authors describe a simpler $\tilde{O}(m^{3/2}\epsilon^{-5/2})$ time algorithm. This simpler algorithm can be converted into an $\tilde{O}(m^{4/3}\epsilon^{-11/3})$ algorithm by removing certain edges of the graph that have large capacity violations in the calculated electrical flows. The final $\tilde{O}(mn^{1/3}\epsilon^{-11/3})$ runtime is a achieved by applying graph smoothing techniques described in [4] to the graph before running the algorithm. This step 'shifts' some of the time dependency from m to n, decreasing the runtime since m may be on the order n^2. Here we do not go into the details of these speed ups but instead describe the simpler $\tilde{O}(m^{3/2}\epsilon^{-5/2})$ algorithm presented in the paper.

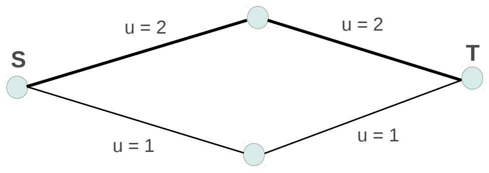
Further, I assume that we are able to calculate the exact electrical flow for any set of resistances. Since the algorithm uses an approximate electrical flow solver, the proofs of the bounds in the papers are (very) slightly more complex.

Throughout the execution of the algorithm, we maintain a vector of edge weights $\boldsymbol{w}$. When calculating an intermediate electrical flow, the resistance of an edge is given by:

$$r_e = \frac{1}{u_e^2}\left(w_e + \frac{\epsilon|\boldsymbol{w}|_1}{3m}\right) \tag{1}$$

where u_e is the capacity of the edge and $|\boldsymbol{w}|_1$ is the sum of edge weights. In the original paper, all edge weights are initialized to 1, causing the initial resistances to be approximately proportional to the inverses of the capacities squared. The initial edge weights do not particularly matter however, with the analysis given in the paper going through as long as they are scaled to sum to m. In practice, it makes sense to set the initial edge weights porportional to the capacities - so that the initial resistances are inversely propositional the the capacities (rather than the capacities squared). In a simple graph, such as the one in Figure 1 below, this will cause initial flows to exactly respect the capacity constraints. In more complex graphs, although the capacity constraints may be violated in the initial electrical flow, having resistances inversely proportional to capacities rather than capacities squared prevents much larger resistances on low capacity edges, causing too little initial flow across these edges.

Figure 1: Setting resistances to 1/u will lead to maximum flow from the start of algorithm



After initializing the edge weights and calculating the initial resistances, we compute the electrical flow of value F from the source s to the sink t. F is an initial guess of the maximum flow value. We will describe below how this guess is updated using a binary search technique. Recall that this electrical flow is the minimum energy flow of value F from s to t. Letting f^* be the actual maximum flow, with value F^* we know that:

$$\begin{aligned}
\mathcal{E}(f^*) &= \sum_e r_e f^*(e)^2 \\
&= \sum_e \frac{1}{u_e^2}\left(w_e + \frac{\epsilon|\boldsymbol{w}|_1}{3m}\right) f^*(e)^2 \\
&= \sum_e \left(w_e + \frac{\epsilon|\boldsymbol{w}|_1}{3m}\right)\left(\frac{f^*(e)}{u_e}\right)^2 \\
&= \sum_e \left(w_e + \frac{\epsilon|\boldsymbol{w}|_1}{3m}\right)(cong_{f^*}(e))^2 \\
&\leq \sum_e \left(w_e + \frac{\epsilon|\boldsymbol{w}|_1}{3m}\right) \\
&= \left(1 + \frac{\epsilon}{3}\right)|\boldsymbol{w}|_1
\end{aligned}$$

since the flow is feasible so $cong_{f^*}(e) \leq 1$ for all edges. Now, if $F \leq F^*$ then the electrical flow f_e, which minimizes the energy over all flows of value F has energy less than or equal to the electrical flow f_e^*, the electrical flow of value F^*. This is because we could just scale down f_e^* to get a flow of value F with lower energy. So, $\mathcal{E}(f_e) \leq \mathcal{E}(f_e^*) \leq \mathcal{E}(f^*) \leq \left(1 + \frac{\epsilon}{3}\right)|\boldsymbol{w}|_1 < (1+\epsilon)|\boldsymbol{w}|_1$.

If the returned electrical flow has energy greater than this bound, then we know that F must be greater than the maximum flow value. So, we update our guess of F (performing binary search between and upper and lower bound for the maximum flow that we compute at the beginning of the algorithm). If the returned electrical flow respects this energy bound, then we can show (see [2] for details) that,

$$\sum_e w_e * cong_{f_e}(e) < (1+\epsilon)|\boldsymbol{w}|_1 \tag{2}$$

and, for all edges e, since $\sum_e r_e * f_e^2 < (1+\epsilon)|\boldsymbol{w}|_1$, we must have:

$$\begin{aligned}
r_e * f_e^2 &< (1+\epsilon)|\boldsymbol{w}|_1 \\
\frac{1}{u_e^2}\left(w_e + \frac{\epsilon|\boldsymbol{w}|_1}{3m}\right) * f_e^2 &< (1+\epsilon)|\boldsymbol{w}|_1 \\
\frac{f_e^2}{u_e^2}\left(\frac{\epsilon|\boldsymbol{w}|_1}{3m}\right) &< (1+\epsilon)|\boldsymbol{w}|_1 \\
cong_{f_e}(e)^2\left(\frac{\epsilon|\boldsymbol{w}|_1}{3m}\right) &< (1+\epsilon)|\boldsymbol{w}|_1 \\
cong_{f_e}(e) &< \sqrt{3m(1+\epsilon)/\epsilon} \\
cong_{f_e}(e) &< 3\sqrt{m/\epsilon}
\end{aligned}$$

if we restrict $\epsilon < 1/2$.

So, our worst case congestion on any edge is:

$$cong_{f_e}(e) < 3\sqrt{m/\epsilon} \tag{3}$$

These bounds, on average congestion and maximum congestion respectively, are key to showing that the average of our electrical flows will eventually converge to a feasible flow of approximately maximum value.

After computing an electrical flow, we update our edge weights using the formula:

$$w_e^i = w_e^{i-1}(1 + \frac{\epsilon}{\rho} cong_{f^i}(e)) \tag{4}$$

where $\rho = 3\sqrt{m/\epsilon}$, is the 'width' of our electrical flow subroutine.

We repeat this process of calculating the electrical flow using our current resistance set and then updating the edge weights (and hence the resistances). After $\frac{2\rho \ln(m)}{\epsilon^2}$ iterations, it can be shown that the appropriately scaled average of all intermediate electrical flows we calculated will have value approximately F and will have no capacity violations. Once calculating such a flow for a value of F, we increase our guess of F and repeat the process. By increasing our guess everytime we compute a feasible flow, and decreasing the guess everytime an electrical flow violates the energy bound discussed above, we can perform a binary search until we have discovered F to the desired level of accuracy.

2 Electrical Flow

As seen in the above description, the fact that the electrical flow minimizes energy over all flows allows us to bound the average congestion and maximum congestion of this flow, and eventually to compose a series of electrical flows into an approximate maximum flow. It is worth discussing a bit more about electrical flows, how they are computed, and how they relate to maximum flows.

As explained above, given a graph G where each edge e has an associated resistance r_e, the electrical flow of value F from a source node s to a sink node t is the flow of value F from s to t that minimizes the energy:

$$\mathcal{E}_r(f) = \sum_e r_e f^2(e)$$

To be a valid flow from s to t, the electrical flow must have no net flow into or out of vertices other than s and t. As the name suggests, the electrical flow is simply the flow that we would see in an electric circuit where each edge is a resistor with value r_e and where s and t are connected with a current source sending F units of current from t back to s (and hence forcing F units of current to flow through the resistor network from s to t.) At first glance the electrical flow seems like it may need to be solved using techniques from optimization theory. However, it is easily found as the solution to a system of linear equations.

2.1 Solution as a System of Equations in the Laplacian.

We first introduce a few matrices that will make the discussion of the electrical flow as the solution to a system of linear equations simpler. Let $n = |V|$ be the number of vertices in our graph and let $m = |E|$ be the number of edges. Further, although we work with undirected graphs, we assign each edge an arbitrary direction. So, each vertex has a set of 'in' edges $E^-(v)$ and a set of 'out' edges $E^+(v)$. The vertex-edge incidence matrix B is the $n \times m$ with

$$B_{v,e} = \begin{cases} 1, & \text{if } e \in E^-(v) \\ -1, & \text{if } e \in E^+(v) \\ 0, & \text{otherwise} \end{cases}$$

Intuitively, B has rows corresponding to vertices and columns corresponding to edges. The column corresponding to $e = (u, v)$ has a zeros everywhere except a 1 at position u and a -1 and position v (where the direction of the edge u to v or v to u is decided arbitrarily.

$$B = \begin{bmatrix} 1 & 1 & \ldots & 0 & 0 & \ldots & 0 & 0 \\ -1 & 0 & \ldots & 1 & 1 & \ldots & 0 & 0 \\ 0 & -1 & \ldots & -1 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & 0 & \ldots & -1 & -1 \end{bmatrix}$$

B clearly contains all the structural information for our graph. However, to incorporate information on edge resistances we use the $m \times m$ diagonal matrix R with $R_{e,e} = r_e$. We often look not at R but at its inverse C. $C_{e,e} = \frac{1}{r_e}$, the conductance of e.

The matrix $L = BCB^T$ is the well known Laplacian matrix of our electrical network. The Laplacian is an $n \times n$ symmetric diagonally dominant matrix. Its diagonal contains the degrees of each vertex weighted by their conductances, and position u, v contains the negative conductance on $e = (u, v)$ if u and v are connected in the graph. Explicitly:

$$L_{u,v} = \begin{cases} \sum_{e \in E(v)} 1/r_e, & \text{if } u = v \\ -1/r_e, & \text{if } e = (u, v) \in V \\ 0, & \text{otherwise} \end{cases}$$

Finally, we introduce the $n \times 1$ vector $\chi_{s,t}$, which is all zeros except with a F at position s (our source node) and a $-F$ at position t (our sink node). $\chi_{s,t}$ is the characteristic vector of an electrical flow of value F. It encapsulates the fact that F units of flow should flow into and out of s and t respectively, and that no net flow should flow into or out of any other vertex (Kirchhoff's first law for electrical circuits). Now, it can easily be seen that any valid flow f of value F from s to t satisfies:

$$Bf = \chi_{s,t}$$

And we can see that:

$$\mathcal{E}_r(f) = \sum_e r_e f^2(e) = (R^{1/2}f)^T \cdot (R^{1/2}f) = ||C^{-1/2}f||^2$$

Now, letting $j = C^{-1/2}f$, we have that solving the electrical flow with value F is the same as solving:

$$\min_f \quad ||j||^2 \quad \text{subject to} \quad BC^{1/2}j = \chi_{s,t}$$

In general, there will be many solutions to $BC^{1/2}j = \chi_{s,t}$. (Except if s and t are disconnected in our graph.) To obtain the solution with the smallest magnitude, we take a solution that is orthogonal to the nullspace of $BC^{1/2}$. If j is not orthogonal to the nullspace of $BC^{1/2}$, then j can be split into two orthogonal vectors $j = j_N + j_R$, where $j_N \in N(BC^{1/2})$, $j_R \in C(A^T)$, and $BC^{1/2}j_R = \chi_{s,t}$.

$||j_N + j_R||^2 \geq ||j_R||^2$, so j_R achieves a lower energy than j. Now, we can obtain the unique j orthogonal to $N(BC^{1/2})$ by using the Moore-Penrose pseudoinverse:

$$j = (BC^{1/2})^+\chi_{s,t}$$

And so, our electrical flow is given by:

$$f = C^{1/2}(BC^{1/2})^+\chi_{s,t}$$

Using the identity: $A^+ = A^T(AA^T)^+$ we get:

$$f = C^{1/2}(C^{1/2}B^T(BCB^T)^+\chi_{s,t}$$
$$f = CB^T(L^+\chi_{s,t})$$

We let the n length vector ϕ be defined as $\phi = L^+\chi_{s,t}$. So $f = CB^T\phi$. ϕ is most intuitively interpreted as a voltage vector for our electrical flow. The flow across any edge $e = (u, v)$ is given by:

$$C_{e,e} * (\phi(u) - \phi(v)) = \frac{\phi(u) - \phi(v)}{r_e}$$

which is the famous Ohm's law, relating voltage and current with the equation $I = \frac{V}{R}$.

So, we can see that solving an electrical flow can be reduced to finding the voltage (potential) vector $\phi = L^+\chi_{s,t}$. Of course, actually computing the puesdo inverse of L is not the fastest way to find ϕ. We explain how we can quickly find ϕ below.

2.2 Quickly Solving Electrical Flows

$\phi = L^{+}\chi_{s,t}$ means that ϕ solves the linear equation $L\phi = \chi_{s,t}$, and that ϕ is orthogonal to the nullspace of L. However, we can see that since the columns (and rows since its symmetric) of L sum to $\mathbf{0}$, the nullspace of L is spanned by the all ones vector $\mathbf{1}$. So not being in the nullspace of L simply means that the voltages ϕ are shifted to have mean 0. Intuitively, we know that shifting all the voltages by a constant amount in an electrical network will not affect the final flow of current. We can see that this is the case by revisting the equation $f = CB^T\phi$. The columns of B^T also sum to 0, so $\mathbf{1}$ is in the nullspace of CB^T. So, $f = CB^T\phi = CB^T(\phi + \boldsymbol{c})$ for any constant vector $\boldsymbol{c}$.

So any solution to $L\phi = \chi_{s,t}$ will be a valid potential vector for our electrical flow. Further, consider fixing $\phi(s) = 1$ and $\phi(t) = -1$. If we solve for a ϕ with $\phi(s) = 1$, $\phi(t) = -1$, and $L\phi = c\chi_{s,t}$ for some constant c, we can clearly simply rescale ϕ so that $L\phi = \chi_{s,t}$. Letting s be the first node and t the last so that the algebra is easier to see, we can convert our system of equations to:

$$\begin{bmatrix} \boldsymbol{e_1} \\ \boldsymbol{L_2} \\ \vdots \\ \boldsymbol{L_{n-1}} \\ -\boldsymbol{e_n} \end{bmatrix} \phi = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix}$$

where e_1 and e_n are the standard unit basis vectors with 1's at positions 1 and n respectively and L_i is the i^{th} row of L. Since this system still enforces the no net outflow condition of vertices other than s and t, a solution to this new system will be a solution to $L\phi = c\chi_{x,t}$ for some c. So such a solution will simply be a scaling of our final solution. Now, we can use row combinations to eliminate the first and last columns of our modified Laplacian matrix. Let c_s be the $n \times 1$ vector with a 1 at position s, $C_{s,u}$ at position u if $e = (u, s) \in E$, and 0 otherwise. Similarly let, c_t be $n \times 1$ with a 1 at position t, $C_{t,u}$ at position u if $e = (u, t) \in E$, and 0 otherwise. Working through the matrix multiplications we get:

$$\begin{bmatrix} \boldsymbol{c_1} \; \boldsymbol{e_2}^T \; \ldots \; \boldsymbol{e_{n-1}}^T \; \boldsymbol{c_t} \end{bmatrix} \begin{bmatrix} \boldsymbol{e_1} \\ \boldsymbol{L_2} \\ \vdots \\ \boldsymbol{L_{n-1}} \\ -\boldsymbol{e_n} \end{bmatrix} \phi = \begin{bmatrix} \boldsymbol{c_1} \; \boldsymbol{e_2}^T \; \ldots \; \boldsymbol{e_{n-1}}^T \; \boldsymbol{c_t} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ -1 \end{bmatrix}$$

$$\begin{bmatrix} & & \boldsymbol{e_1} & & \\ 0 & L_{2,2} & \ldots & L_{2,n-1} & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & L_{n-1,2} & \ldots & L_{n-1,n-1} & 0 \\ & & \boldsymbol{e_n} & & \end{bmatrix} \phi = \boldsymbol{c_s} - \boldsymbol{c_t}$$

Since the first and last variables of this equation are determined we can simply strip off the first and last columns of the matrix and the first and last elements of the vectors to the get linear equation:

$$L_{inner} \begin{bmatrix} \phi(2) \\ \vdots \\ \phi(n-1) \end{bmatrix} = \begin{bmatrix} (\boldsymbol{c}-\boldsymbol{t})(2) \\ \vdots \\ (\boldsymbol{c}-\boldsymbol{t})(n-1) \end{bmatrix}$$

where L_{inner} is the 'inner Laplacian' - L without its first and last rows and columns. It turns out that L_{inner} *is positive definite.* It is well known that L, when edge weights are positive, is positive semi-definite since its quadratic form gives the smoothness of a function over a graph:

$$\mathbf{x}^T \mathbf{L}\mathbf{x} = \frac{1}{2} \sum_{e=(u,v)\in E} C_{u,v}(\mathbf{x}(u) - \mathbf{x}(v))^2 \geq 0$$

The inner Laplacian gives the smoothness over $G - \{s,t\}$ so is still positive semi-definite. In fact, it has no nullspace, since it has the same row space as the modified Laplacian with its first and last rows and columns reduced to unit vectors, except its row space does not include e_1 and e_n. Since this modified Laplacian had full rank (we fixed $\phi(s)$ and $\phi(t)$ to eliminate the all-ones vector nullspace), the inner Laplacian (with two fewer rows), must also have full rank.

The positive definiteness of the inner Laplacian allows us to quickly solve the above linear equation using a variety of techniques. For smaller matrices a Cholesky decomposition can be used. For large matrices, we can use the iterative conjugate gradient method. When combined with preconditioning (an incomplete Cholesky preconditioner in my `MATLAB` implementation), this allows us to quickly solve electrical flows even over massive graphs with tens of thousands of nodes and hundreds of thousands of edges.

Of course, the run time bounds given in the original paper are based off using the current fastest known algorithm for approximately solving Laplacian systems. This algorithm will return a value for ϕ satisfying $||\phi - L^{+}\chi_{s,t}||_L < \epsilon||L^{+}\chi_{s,t}||_L$ in time $\tilde{O}(m\text{log}n + n\text{log}^2 n * \log(1/\epsilon))$ where $||x||_A^2 = x^T Ax$. [5]. However, because this algorithm does not currently have fast or stable implementations, we use the more standard techniques described above in our testing of the maximum flow algorithm.

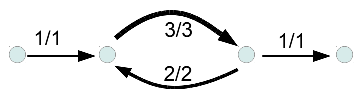
2.3 Maximum Flows as Electrical Flows

It is worth noting a couple interesting properties about electrical flows and their relationship to maximum flows.

Recall from above that we find an electrical flow f with the equation $f = CB^T\phi$. So, f is in the row space of BC, and so is orthogonal to all vectors in its nullspace. Setting each resistance value to 1, $BCf = Bf = \chi$. So, any flow in the nullspace of B is one that leads to $\chi = \mathbf{0}$, so no net flow into any vertex. Such a flow is called a 'circulation'. So, electrical flows are orthogonal to circulations. If the resistances are not all 1, then an electrical flow is orthogonal to capacitance weighted circulations.

Intuitively, when finding a maximum flow we would like the avoid circulations since they simply congest edges without increasing the flow from s to t. Of course not all maximum flows avoid circulations. When there is excess capacity in the graph, there may be many maximum flows that have circulatory components.

Figure 2: A circulating maximum flow that could not be an electrical flow with positive resistances.



However, we can see that there always exists at least some maximum flow that can be written as an electrical flow. In fact, if we fix a set of voltages satisfying a few basic properties, we can always find a set of resistances whose electrical flow will be a maximum flow and whose vertex potentials will be equal to the fixed set of voltages.

Theorem 2.1. *There always exists a maximum flow that can be written as an electrical flow.*

Proof. Consider any maximum flow f^* over the graph G. Now, we would like to eliminate all circulations from f^*. If f^* has a cycle of vertices $v_1, v_2, ...v_k$ such that the flows on edges $(v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k), (v_k, v_1)$ all have the same sign (either all strictly positive or strictly negative - but we will assume positive without loss of generality), then set $f(e_{min})$ to be the minimum flow on one of these edges, e_{min}. If we then subtract $f(e_{min})$ from the flow on all edges in the cycle, we see that the total flow into and out of every vertex remains the same, so we still have a maximum flow. Further, e_{min} now has flow 0 across it, while the flow on all other vertices in the cycle has remained positive. So, we have removed the cycle of strictly positive flow. And, since we have not flipped the sign of the flow on any edge, we have not created any new cycles with either strictly positive or strictly negative flow. By repeating this process we can arrive by construction at some flow f^* that contains no cycles of edges all with either strictly positive or strictly negative flow.

Now, we can assign voltages ϕ and resistances $\boldsymbol{r}$ to the vertices and edges of G so that f^* is the electrical flow over G with resistances $\boldsymbol{r}$ and leads to the potential vector ϕ. We start by assigning s to have an arbitrary voltage, possibly 1. We then consider all neighbors of s (all $v \in N(s)$). If f^* has a positive flow going from s to some neighbor v, we assign v to have a voltage lower than s. We then look at the interactions amongst the set of neighbors $N(s)$. Whenever one neighbor v_1 has positive flow going to another neighbor v_2, we decrement the voltage of v_2 to be lower than the voltage of v_1. Since we have removed all positive flow edge cycles from f^*, this process of decrementing voltages will eventually terminate. We continue on, assigning voltages to the neighbors of all members of $N(s)$, at each stage decrementing voltages as necessary so that all positive flows in f^* are from vertices with higher potentials to vertices with lower potentials. Upon completing this process we consider each edge $e = (u, v)$ of G. If f^* sends a flow of value $F > 0$ from u to v, then our above process guarantees that $\phi(u) > \phi(v)$ and we can set $r_e = \frac{\phi(u)-\phi(v)}{F}$. If instead f^* sends a positive flow from v to u then our above process guarantees that $\phi(v) > \phi(u)$ and we can set $r_e = \frac{\phi(v)-\phi(u)}{F}$. If no flow moves along e then we can set $r_e = \infty$. Or, using a slightly more complex voltage assignment process, could have set $\phi(u) = \phi(v)$, so r_e can take any value.

Essentially what we have done above is found a maximum flow with no circulations - i.e. a maximum flow that could be made orthogonal to all $\tilde{f}$ with $BC\tilde{f} = 0$. Such an maximum flow can be written as an electrical flow. $\square$

Of course, writing a given maximum flow as an electrical flow is a much easier task than actually finding the proper set of resistances that will lead to an electrical flow that is a maximum flow. In the algorithm studied such a set of resistances may never actually be found. The final approximate maximum flow is obtained by averaging a large set of electrical flows, all calculated over different resistance sets. Empirically however, it does seem that the algorithm always converges to a set of resistances that gives a feasible approximate maximum flow. As will be discussed below, proving that this convergence always occurs would be an important step in understanding this algorithm.

2.4 A Difference in Norms

One interesting way to look at the difference between calculating electrical flows and calculating maximum flows is that both calculations require minimizing the magnitude of the energy vector of a flow over the graph, but use different vector norms when calculating this magnitude. As explained above, letting F be the value of the maximum flow on our graph, the electrical flow of value F minimizes the energy $\mathcal{E}_r(f) = \sum_e r_e f_e^2 = \|\sqrt{\mathbf{r_e}}\mathbf{f_e}\|^2$ where $\sqrt{\mathbf{r_e}}\mathbf{f_e}$ is the m length vector with value $\sqrt{r_e} * f_e$ at position e. If r_e is set proportional to $1/u_e^2$, then calculating the electric flow corresponds to minimizing $\|\mathbf{f_e}/\mathbf{u_e}\|_2$ (minimizing the 2-norm).

Now, a maximum flow on the graph must have at least 1 edge saturated to full capacity (or else we could scale up the flow by a constant to get a higher flow value.) However, it has no edges flowing over capacity. So, letting $r_e = 1/u_e$, a maximum flow is a flow of value F that minimizes the value $\sum_e (r_e * f_e)^\infty = \|\mathbf{f_e}/\mathbf{u_e}\|_\infty$. A maximum flow minimizes this value to 1. Any flow of value F with capacity constraints violated would have this value go to infinity. Any flow where this value is 0 must have all edges unsaturated, and so cannot be a maximum flow, and so cannot have value F.

If we had an efficient algorithm to minimize the infinity norm of $\mathbf{r_e}\mathbf{f_e}$ over all valid flows, we could set $r_e = 1/u_e$ and compute the maximum flow using this algorithm. However, we do not have such an algorithm. Instead we use the fact that linear algebra allows us to efficiently minimize the 2-norm of vectors and use a series of 2-norm minimizations (in calculating the electrical flows) to eventually lead to the approximate infinity norm minimization.

The runtime of our algorithm is inherently limited by the gap between the 2-norm and the infinity norm. As explained above, with the resistance scheme used in the algorithm, an electrical flow with value F will never violate the capacity of an edge by a factor greater than $3\sqrt{m/\epsilon}$. If we used the appropriate resistance values and calculated a general p-electrical flow, a flow minimizing the p-norm of the energy vector $\boldsymbol{r_e f_e}$, then the capacity would not be violated by more than a factor of $O((\frac{m}{\epsilon})^{1/p})$. This would allow us to iterate on the order of $(\frac{m}{\epsilon})^{1/p}\left(\frac{\ln(m)}{\epsilon^2}\right)$ times rather than $(\frac{m}{\epsilon})^{1/2}\left(\frac{\ln(m)}{\epsilon^2}\right)$ times.

Of course, while linear algebraic techniques give many techniques for minimizing vector 2-norms, much less is known about minimizing higher order norms. It would be an interesting line of research to see if looking at higher order norm minimization could somehow lead to an improved maximum flow algorithm (or improved infinity norm minimizations in general). Or to see if there is some limit to the efficiency of higher order norm minimization algorithms that is related to the efficiency of 2-norm minimization algorithms.

3 Removing Binary Search with Flow Scaling

As described above, in the original algorithm being studied, the maximum flow value was found through a process of binary search. Initially upper and lower bounds on the maximum flow value are calculated (using a variety of possible techniques). Call these bounds f_l and f_u. At each step of the binary search the algorithm attempts to find a feasible flow with inputed value $F \in [f_l, f_u]$. Letting F^* be the actual maximum flow value, if $F \leq F^*$, then, as described above, each intermediate electrical flow will respect the $(1+\epsilon)|\boldsymbol{w}|_1$ energy bound. So, the algorithm will always return a feasible flow with value F. If $F > F^*$, then any flow of value F will be infeasible. So, when running the algorithm, some intermediate electrical flow must break the energy bound (or else, the algorithm would produce a feasible flow of value F, which is a contradiction). Note: Technically, if we have $F^* \leq F \leq \frac{1}{1-O(\epsilon)}F^*$ then the algorithm may return a feasible flow with value $< F^*$ and no intermediate electrical flow failure.

However, since the algorithm will always indicate failure if $F > F^*$ by a wide enough margin because an electrical flow will violate the energy bound, and will always return a feasible flow of value F if $F \leq F^*$, we can perform a binary search for F in the range $[f_l, f_u]$. This search does not increase the asymptotic runtime of the algorithm as it only adds a multiplicative factor logarithmic in m and ϵ. However, in practice, performing the binary search can significantly increase runtime and seems unnecessary.

Below I will explain how the binary search can be avoided for the simpler

$\tilde{O}(m^{3/2}e^{-5/2})$ algorithm through the use of 'flow scaling'. Unfortunately, despite some effort, I have not yet been able to prove that we can avoid binary search when using the faster $\tilde{O}(m^{4/3}\epsilon^{-11/3})$ version of the algorithm with edge cutting.

At each iteration of the algorithm the electrical flow over the current resistance set is returned. Normally if this flow has energy $> (1+\epsilon)|\boldsymbol{w}|_1$, we will halt iteration because we will know that $F > F^*$. However, letting $\mathcal{E}(f)$ be the energy of the returned flow, then by multiplying the flow value on each edge by the factor: $\sqrt{\frac{(1+\epsilon)|\boldsymbol{w}|_1}{\mathcal{E}(f)}}$, we obtain a valid electrical flow that is a scaling of the returned flow but has energy exactly equal to $(1+\epsilon)|\boldsymbol{w}|_1$. We can in fact solve the initial electric flow with any flow value we choose - for example, we can always force 1 unit of flow from s to t. Since we then scale up to the energy bound, the initial value of the computed electrical flow is irrelevant.

Theorem 3.1. *If at each step of the algorithm described in [2] we solve an electrical flow of value* 1 *on the graph and then set* $f = \left(\sqrt{\frac{(1+\epsilon)|\boldsymbol{w}|_1}{\mathcal{E}(f)}}\right) * f$ *and update weights as described, the algorithm will converge so that* $\bar{f} = \frac{(1-\epsilon)^2}{(1+\epsilon)N}(\sum_i f^i)$ *is a feasible, approximate maximum flow.*

Proof. The fact that each scaled flow respects the $(1+\epsilon)|\boldsymbol{w}|_1$ energy bound means that the congestion bounds shown in equations (2) and (3) still hold. So, the proof that the final average of electrical flows is feasible goes through exactly as presented in the paper.

Further, we have that the scaled value of the electrical flow is at least as large as the true maximum flow value F^*. If the value were any less, then an electrical flow of value F^* could be found by scaling up f. However, the scaled up version of f would have energy $> (1+\epsilon)|\boldsymbol{w}|_1$. This is a contradiction since this flow must have energy $\le \mathcal{E}(f^*) \le (1+\epsilon)|\boldsymbol{w}|_1$.

Now, we show that when simply scaling the electrical flows instead of failing, Lemma 3.4 and Lemma 3.5 of the original paper still hold so the multiplicative weights update technique still converges to a feasible average flow. Lemma 3.4 states that for any $i \ge 0$

$$\mu_{i+1} \le \mu_i exp\left(\frac{(1+\epsilon)\epsilon}{\rho}\right)$$

where $\rho = 3\sqrt{m/\epsilon}$ is the width of the electrical flow oracle. Since, as explained above, the average congestion bound: $\sum_e w_e^i cong_{f^{i+1}}(e) \le (1+\epsilon)|\boldsymbol{w^i}|_1$ still holds, we can follow the same proof in the original paper:

$$\begin{aligned}\mu_{i+1} &= \sum_e w_e^{i+1} = \sum_e w_e^i\left(1 + \frac{\epsilon}{\rho} cong_{f^{i+1}}(e)\right)\\ &= \sum_e w_e^i + \frac{\epsilon}{\rho}\sum_e w_e^i cong_{f^{i+1}}(e) \le \mu_i + \frac{(1+\epsilon)\epsilon}{\rho}|\boldsymbol{w}^i|_1\end{aligned}$$

Again following the analysis of the paper this gives Lemma 3.4: $\mu_{i+1} \le \mu_i exp\left(\frac{(1+\epsilon)\epsilon}{\rho}\right)$.

In particular this gives $\mu_N \leq m * exp\left(\frac{(1+\epsilon)\epsilon}{\rho} N\right)$.

We can also easily see that Lemma 3.5 holds, since it only uses the fact that in each electrical flow, for each edge e, $cong_{f^i}(e) \leq \rho$, which holds after we scale the flow to the proper energy threshold. Following the analysis of the paper, we can use this maximum congestion bound to show that

$$w_e^i \geq exp\left(\frac{(1-\epsilon)\epsilon}{\rho} \sum_{j=1}^{i} cong_{f^j}(e)\right)$$

Setting $i = N$ gives

$$w_e^N \geq exp\left(\frac{(1+\epsilon)\epsilon N}{(1-\epsilon)\rho} cong_{\bar{f}}(e)\right)$$

And again following the analysis of the paper, we combine the two lemmas above to get that for each edge e:

$$\begin{aligned} m * exp\left(\frac{(1+\epsilon)\epsilon}{\rho} N\right) \geq \mu_N \geq w_e^N &\geq exp\left(\frac{(1+\epsilon)\epsilon N}{(1-\epsilon)\rho} cong_{\bar{f}}(e)\right) \\ \ln(m) + \left(\frac{(1+\epsilon)\epsilon}{\rho} N\right) &\geq \left(\frac{(1+\epsilon)\epsilon N}{(1-\epsilon)\rho} cong_{\bar{f}}(e)\right) \\ cong_{\bar{f}}(e) &\leq 1 - \epsilon + \frac{(1-\epsilon)\rho \ln(m)}{(1+\epsilon)\epsilon N} \leq 1 \end{aligned}$$

So, the moral of the above algebra is that, as long as each electrical flow is scaled to have energy $(1+\epsilon)|\boldsymbol{w}|_1$ (where the weights are those used to give the resistances used to calculate the flow), then the original bounds shown in the paper still hold. So, the average of the electrical flows is feasible. Since each scaled electrical flow has value $\geq F$, this average approximates the maximum flow within order ϵ since $\bar{f} = \frac{(1-\epsilon)^2}{(1+\epsilon)N}(\sum_i f^i)$, where f^i is the ith scaled electrical flow. $\square$

3.1 Flow Scaling and Convergence on Maximum Flow

As discussed in more detail below, an open question about the algorithm being studied is whether the averaging of intermediate electrical flows is necessary. In practice it seems that the final electrical flow is always a feasible approximate maximum flow, making averaging unnecessary. However, we currently have no proof that the electrical flows don't have some sort of oscillatory behavior, with different edges over capacity in each flow despite having a feasible average flow.

The modification of the algorithm to not use binary search gives at least a different view of the problem of determining whether the electrical flows do in fact converge on a maximum flow without averaging. All scaled electrical flows have flow value $\geq F^*$ as explained above. Yet, since $\bar{f}$ is feasible, it must have flow value $\leq F^*$. $\bar{f} = \frac{(1-\epsilon)^2}{(1+\epsilon)N} \sum_{i=1}^{N} f^i$. Now, empirically, it seems that if we record the flow values of the scaled electric flows in our non-binary search algorithm, these values decrease throughout the course of the algorithm. This means that, if we solve the electrical flows always with a constant value before scaling, every time we update our edge weights, we are increasing the ratio:

$$\frac{\mathcal{E}(f)}{|\boldsymbol{w}|_1}$$

where $\mathcal{E}(f)$ is the energy of the electrical flow over the graph with value F, the constant throughput value we use. If this ratio is in fact increasing monotonically, and so the scaled flow values are decreasing monotonically, in order to have $\bar{f}$ with value $\leq F^*$ we must have the electrical flows converging to have scaled value near F^*. Specifically, letting f^N be the final electrical flow and F^N be the final scaled electrical flow value, we would have:

$$\begin{aligned}
\bar{F} &= \frac{(1-\epsilon)^2}{(1+\epsilon)N} \sum_{i=1}^{N} F^i \\
\bar{F} &\geq \frac{(1-\epsilon)^2}{(1+\epsilon)N} N * F^N \\
F^* &\geq \frac{(1-\epsilon)^2}{(1+\epsilon)} F^N \\
F^N &\leq \frac{(1+\epsilon)}{(1-\epsilon)^2} F^*
\end{aligned}$$

Further, the scaled f^N has $\mathcal{E}(f^N) = (1+\epsilon)|\boldsymbol{w^N}|_1 \geq \mathcal{E}(f^*)$. And, $\mathcal{E}(f^*) \geq \mathcal{E}(f)$, where f is the electrical flow with value F^*. Additionally, $\mathcal{E}(\frac{(1+\epsilon)}{(1-\epsilon)^2} f) \geq \mathcal{E}(f^N)$ since it has at least as high a flow value. These facts give us:

$$\begin{aligned}
\left(\frac{(1+\epsilon)}{(1-\epsilon)^2}\right)^2 \mathcal{E}(f) &\geq \mathcal{E}(f^N) \\
\frac{(1+\epsilon)^2}{(1-\epsilon)^4} \mathcal{E}(f) &\geq \mathcal{E}(f^*) \\
\frac{(1+\epsilon)^2}{(1-\epsilon)^4} \mathcal{E}(f) &\geq \mathcal{E}(f^*) \geq \mathcal{E}(f)
\end{aligned}$$

where the above energies are calculated using the final resistance set r^N. This means that, with our final resistance set, the maximum flow has nearly minimal energy (Within the factor $\frac{(1+\epsilon)^2}{(1-\epsilon)^4}$.) Perhaps this means that the maximum flow is also nearly an electrical flow given these resistances.

Of course our assumption that the scaled flow values decrease over the course of the algorithm is based only off empirical findings. Proving this fact is still open. It is possible (and was the case in my initial non-binary search implementation) to force this behavior by recording a 'flow goal value'. Whenever an electrical flow breaks the energy barrier, it is scaled down to have energy $\mathcal{E}(f) = (1+\epsilon)|\boldsymbol{w}|_1$. The flow value of this scaled flow becomes the new flow goal. Future electrical flows are calculated to have this same goal and scaled down further if they again break the energy barrier. In this way, we maintain a constantly decreasing goal, which is similar to a dual upper bound variable used in some interior point algorithms. However, it may be that in this method the final electrical flow, which is scaled to have value equal to the flow goal, has energy much lower than $(1+\epsilon)|\boldsymbol{w}^{\boldsymbol{N}}|_1$. So, it is not clear that the maximum flow has nearly minimal energy under the final resistance set.

Even given the monotonic decrease of the scaled flow values, it is unclear whether the approximate energy optimality of the maximum flow means that this flow is close to an electrical flow. More work would have to be done to determine how being close to a flow energy-wise translates to being close flow-wise.

The reason that we cannot so easily prove that the flow scaling approach works when using the $\tilde{O}(m^{4/3}\epsilon^{-11/3})$ time algorithm is that in this faster algorithm, anytime an edge has congestion greater than $\rho = \tilde{O}(m^{1/3}/\epsilon)$, the edge is cut from the graph. In this way, we effectively reduce ρ from depending on $m^{1/2}$ to depending on $m^{1/3}$ without changing the operation of the algorithm. It is possible to bound the total capacity of cut edges so we know that the final returned flow is still approximately maximal. Unfortunately, this bound relies on one very simple fact that we lose when taking the flow scaling approach - that we never flow more than F^* units across a single edge in an electrical flow. In the original algorithm this is ensured because if the algorithm runs to completion then F, the value of the intermediate electrical flows, is $\leq F^*$. So, it is never necessary to flow more than F units across an edge in an electrical flow. However, with flow scaling, we may scale an electrical flow to have value greater than F^* and so to possibly have more than F^* units of flow moving across a single edge. If we could better understand how the electrical flows actually evolve, rather than just the properties of their average, then it seems that we could bound the flow across a single edge in the graph after a certain number of iterations. This would then allow us to bound the capacity of cut edges and show that the faster algorithm could work without binary search.

3.2 Relationship to Interior Point Linear Programming Algorithms

While working on this project, I spent a fair amount of time learning about interior point algorithms for linear programming and seeing if I could connect them to the algorithm being studied. I made little real progress in making connections between the two and was eventually sidetracked into looking at possible oscillatory behavior of the intermediate electrical flows and eventually at the connection between maximum flow and thermistor circuits. In this section I just briefly present how the algorithm being studied seems to relate to interior point methods. Continuing to look at the connections between the two could be an interesting area of future work.

Letting B be the vertex edge incidence matrix defined above, and assuming that our first vertex is the source node and the last vertex the sink, we can formulate the undirected maximum flow problem as a standard form linear program. We convert the undirected problem into a directed problem, where each original edge (u, v) now corresponds to two directed edges with the same capacity flowing from u to v and vice versa. In this way, our flow vector will only have positive values, so will obey the non-negativity constraint used for standard form linear programs. Let B^2 be the $n \times 2m$ vertex-edge matrix for the new directed graph. Let B^2_{inner} be B^2 with the first and last rows removed (those corresponding to the source and the sink). Let B^2_1 be the first row vector of B^2 - which gives the in and out edges for the source vertex.

Consider some edge e in our undirected graph. If in some maximum flow e is meant to have flow $f(e) \leq u_e$ across it there are many ways to achieve the equivalent of this flow in our directed graph. We can send $f(e)$ from u to v and nothing back from v to u. Or, since $f(e) \leq u_e$ so $u_e - f(e) = c \geq 0$, we could send $f(e) + \frac{c}{2}$ from u to v and send $\frac{c}{2}$ back from v to u. This would achieve a total flow of $f(e)$ from u to v, while still maintaining feasibility over our two directed edges.

In order to simplify our linear program, we will always use the second approach above. This ensures that in any solution, the total flow along the directed edges corresponding to e is:

$$\left(f(e) + \frac{u_e - f(e)}{2}\right) + \left(\frac{u_e - f(e)}{2}\right) = u_e$$

Now, letting the directed edges corresponding to each undirected edge e be e_1 and e_2, and calling our new edge set $E_1 \cup E_2$, we use f_1 to denote the vector of positive flows across the edges in E_1 and f_2 to denote the vector of positive flows across the edges in E_2. We can finally set up our linear program as:

$$\min_{\boldsymbol{f_i}} \qquad -B^2_1 \cdot \begin{bmatrix} \boldsymbol{f_1} \\ \boldsymbol{f_2} \end{bmatrix} \qquad \text{s.t.} \qquad \begin{bmatrix} B^2_{inner} \\ I_m \; I_m \end{bmatrix} \begin{bmatrix} \boldsymbol{f_1} \\ \boldsymbol{f_2} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{u} \end{bmatrix}$$

$$\text{and } \boldsymbol{f_i} \geq 0$$

Where I_m is the $m \times m$ identity matrix. This linear program can be solving using one of many standard linear programming algorithms - including interior point algorithms. Although the algorithm studied in this paper is specifically designed to solve maximum flow, it seems to be related to these generic interior point algorithms. Perhaps by looking at the connections between the two it is possible to build a better understanding of the algorithm's behavior. In the most general terms, an interior point algorithm solves a linear program by starting at a feasible solution of the linear program. The algorithm then makes steps within the feasible region of the program, eventually converging to the optimal solution. It is worth noting that our algorithm is *not* an interior point algorithm (or at least has not been proven to be). The intermediate electrical flows that we compute at each iteration may be infeasible. It is in fact possible (although not observed in practice) that no electrical flow we compute over the course of the algorithm will ever be feasible. However, more important to look at is the cumulative average of the electrical flows which at step k can be defined as:

$$\bar{f}^k = \frac{(1-\epsilon)^2}{(1+\epsilon)N}(\sum_{i=1}^{k} f^i)$$

As k goes to n, $\bar{f}^k$ converges to an approximate maximum flow. However, $\bar{f}^k$ may be infeasible during the execution of the algorithm. Specifically, if the flow along an edge switches direction as we update resistance, then even though $\bar{f}^k$ is scaled down significantly by the $\frac{(1-\epsilon)^2}{(1+\epsilon)N}$ factor, it could be infeasible at some step. Of course it seems highly unlikely that this would occur - and showing that $\bar{f}^k$ remains feasible would be another goal in better understanding the behavior of this algorithm.

Despite the fact that our algorithm is not strictly an interior point algorithm is does share some of the higher level design of these algorithms. One way of implementing the algorithm without binary search is to maintain a 'flow goal' g during the course of the algorithm. g is initialized to be an upper bound of the maximum flow value. Whenever an electrical flow breaks the energy threshold $(1+\epsilon)|\boldsymbol{w}|_1$, it is scaled down to meet the threshold. g is then reset to the flow value of this scaled down flow. In this way, g acts as a constantly decreasing upper bound on the maximum flow. As described in the previous section, in order for $\bar{f}$ to be feasible this upper bound must converge to be near the actual maximum flow.

The use of a flow goal seems very similar to the use of an upper bound for the optimal value of the linear program in potential reduction interior point methods, such as the Affine Potential Reduction algorithm described in [1]. In potential reduction algorithms, the primal potential function:

$$f(x,z) = q\ln(c^T x - z) - \sum_{i=1}^{n} \ln(x_i)$$

is often considered. x is the current interior feasible solution to the linear program. z is an upper bound on the optimal value of the linear program $c^T x^*$ (where x^* is the optimal feasible solution - f^* in the maximum flow case). These algorithms work by iteratively updating x and z in a way that guarantees a reduction in $f(x,z)$. As z converges towards $c^T x$ (so x converges towards being optimal) the $q\ln(c^T x - z)$ term goes to $-\infty$. The $-\sum_{i=1}^{n} \ln(x_i)$ goes to infinity on the boundary of our feasible space (when $x_i = 0$ for any i), so 'pushes' our intermediate values of x towards the interior of this region. (Note: a more detailed explanation of why minimizing the barrier function leads to solving our linear program is given in [11].)

One interesting question is whether there is some potential function that is decremented at each step of our algorithm. The use of such a potential function could help prove that the intermediate electrical flows actually converge to an approximate maximum flow and avoid the use of $\bar{f}$. If for any intermediate electrical flow we define $v(f)$ to be the flow value of f once f is scaled to be feasible, then it is still unclear that our flow goal g approaches $v(f^i)$ as we iterate. As the oscillations discussed in the next section demonstrate, there do clearly exist graphs on which $v(f)$ does not increase monotonically. So, it may be that convergence only occurs after an initial 'burn in' period during which unstable initial conditions settle out.

Another interesting question to explore is whether graphs that cause poor performance of our algorithm also cause poor performance of traditional interior point algorithms. Identifying a relationship here could help us understand how the algorithms are fundamentally similar or different. Of course, it can be difficult to find graphs on which our algorithm 'behaves poorly'. Perhaps it would be useful to look at combinations of the oscillating graphs described below and the graphs that nearly reach the ρ bound on maximum congestion as described in [2].

4 Oscillatory behavior of intermediate electrical flows

As mentioned earlier, a major open question about the current algorithm is whether the averaging of all intermediate electrical flows is neccesary. Empirically, it seems that the edge resistances always converge to a steady configuration that gives an electrical flow that is itself an approximate maximum flow. However, we have no proof of this fact. It is possible that having a certain set of edges under capacity in an electrical flow will cause other edges to be over capacity, and that our algorithm will never produce an electrical flow that is feasible.

In trying to prove that the electrical flows always converge to a feasible maximum flow, I attempted to construct graphs in which edge congestions oscillated. It is in fact very simple to produce oscillations using resistor chains. In a graph, consider replacing an edge with capacity u_e with a chain of edges, each with capacity u_e. This replacement does not effect the maximum flow of the graph. However, if we run our algorithm on the graph, we initialize resistances to be inversely proportional to capacities. So in the original graph our initial r_e is approximately $\frac{1}{u_e}$. The resistance on each edge in the new graph is set in the same way. However, since resistors in series are additive, this means that the total resistance across the chain of edges is $l * \frac{1}{u_e}$, where l is the number of edges in the chain.

In fact, the possibility of edge chains demostrates why the initial weights we assign to the edges, while important in practice, do not affect the overall runtime of the algorithm. By replacing each edge in the graph with chains of varying lengths we could force any arbitrary initial resistance set (considering the total resistances across edge chains), and so any arbitrary initial flow. Since, in order for the conservation of flow to hold, each edge in a chain must flow the same amount, the resistances on the chains in the graph will be updated exactly how the resistances of a single edge in the original graph would be updated. So, solving the maximum flow on the chain graph will essentially simulate solving the maximum flow on the original graph, except with a different initial edge weight vector.

If a chain is long and so $l(\frac{1}{u_e})$ is very high, flow may stay too low on the edges in the chain for a number of iterations of the algorithm. By combining chains of different lengths we can easily generate occilatory behavior. In the graph in Figure 3 below, the source and sink nodes each have three outgoing edges connecting to three separate paths - chains of length 100, 10, and 1. All edges have capacity 1, so the maximum flow is 3 and is achieved by saturating all edges fully. However, the chain of length 100 will initially have a much higher total resistance, so will start with edges flowing far below capacity. The edge in the chain of length 1 will correspondingly flow over capacity in our initial electrical flows. Eventually, as we increment the resistance on the short chain relative to the resistances on the long chain, the flow values will converge to 1. Interestingly, the edges in the middle chain of length 10 will initially start below capacity. As we increment the resistance on the length 1 chain, the length 10 chain will absorb some of the flow being taken off the shortest chain. Its edges will begin to flow over capacity before its own resistance is incremented sufficiently and all edges converge to flowing 1 unit. Figure 4 shows how the flows evolve over the course of the algorithm. The decreasing line is the flow along the edge in the length 1 chain, and the increasing is the flow along the edges in the length 100 chain. The middle line shows how the edges in the length 10 chain start below capacity, go above capacity, and eventually drop to full saturation at $f = 1$.

Figure 3: Basic chain graph with chains of length 100, 10, and 1

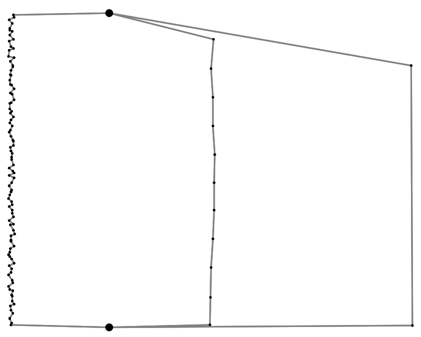
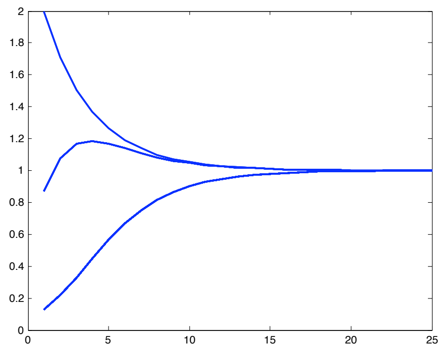
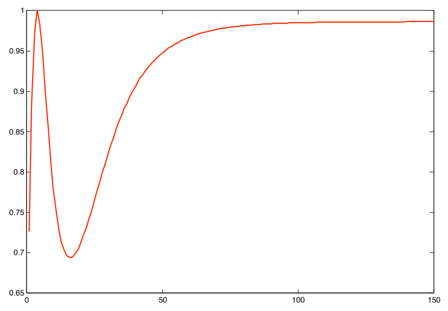


Figure 4: Edge flows in initial iterations of algorithm.



While this very simple graph only demonstrates a flow oscillation with one peak, it is possible to construct graphs in which the flow oscillates multiple times. By simply taking the above graph G and replacing one of the edges in the middle chain by a full copy of G, with capacities scaled down to $1/3$, we get another graph with maxflow value 3. If we look at the flow along an edge in the middle chain of the copy of G that was inserted, we see that this flow oscillates twice, moving from around 70% capacity up to full capacity, back down to 70% capacity, and finally back up to full capacity in the steady state.

Figure 5: Double oscillation on chain graph



Of course, the above oscillations are not sustained. They are reminiscent of damped harmonic oscillators - displaying oscillatory behavior initially due to a specific starting condition but eventually settling down to a steady state. It remains open whether sustained flow oscillations could occur, forcing the use of the averaging technique used in the algorithm.

5 Maximum flow though thermistor circuits

In considering whether or not flow oscillations are possible in the current algorithm it seems relevant to consider a continuous analog to our iterative algorithm. The study of oscillatory behavior is common in work with differential equations and continuous time phenomena such as the study of the behavior of electrical circuits containing capacitors, inductors, and other elements.

Consider starting with the algorithm described in [2], but setting ϵ to an infinitesimally small step size. So we have, going back to equation (4), $w_e(t+\epsilon) = w_e(t)(1+\frac{\epsilon}{\rho}cong_e(t))$. $\rho = 3\sqrt{m/e}$ is inversely dependent on ϵ so goes to infinity as ϵ goes to 0. So, the evolution of our edge weights can then be described by:

$$\frac{dw_e}{dt} = cong_e * w_e$$
$$\frac{dw_e}{dt} = \frac{f_e}{u_e} w_e \tag{5}$$

Further, as ϵ goes to 0, we have $r_e = \frac{w_e}{u_e^2}$.

It seems that starting from these equations it may be feasible to study the evolution of the electric flow as we update the weights and resistances. We know that for a given matrix A, $(A+\epsilon X)^{-1} = A^{-1} - \epsilon A^{-1} X A^{-1}$. This formula should allow us to study how the inverse Laplacian (or technically, the inverse of the positive definite inner Laplacian) changes with the changing resistance values. This would show us to see how flow values change over time.

One unnatural fact about these equations is that since dw_e/dt is always a positive multiple of w_e the weights and resistances all grow exponentially. This cause the conductance values, which are the weights of the Laplacian matrix, to move towards 0.

A seemingly more natural set of differential equations is found by looking at the behavior of a relatively common electronic circuit element - a thermistor. A thermistor is a resistor whose resistance value is heavily affected by its temperature. Using the basic equations introduced in [8] and [10], letting T_e be the temperature of the thermistor and r_e be its resistance, we have $r_e = T_e\alpha_e$, where α_e is a constant temperature coefficient of resistance. If we model an edge of our graph with a thermistor and set $\alpha_e = 1/u_e^2$, then we have $r_e = \frac{T_e}{u_e^2}$. Considering the temperature to be the edge weight, this is the same relationship we have in the original algorithm. Now, the evolution of the temperature of the thermistor over time is given by:

$$\frac{dT_e}{dt} = \left(f_e^2 \alpha_e T_e\right) + d(T_a - T_e) \tag{6}$$

where T_a is an outside ambient temperature and d is the rate of heat dissipation to the environment. The term $f_e^2\alpha_e T_e$ derives from the fact that power delivered to the thermistor at a given point in time is $v(t) * i(t) = v_e * f_e = r_e * f_e^2 = f_e^2\alpha_e T_e$.

Again setting $\alpha_e = 1/u_e^2$, and simplifying the equation by setting $T_a = 0$ and $d = 1$ we have:

$$\frac{dT_e}{dt} = \left(\frac{f_e^2}{u_e^2} - 1\right) T_e \tag{7}$$

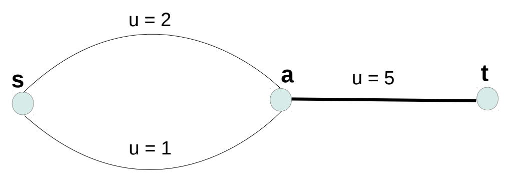
This equation is really pretty similar to the equation for dw_e/dt in the original algorithm. Notably, the congestion is squared and recentered with the -1, so we do not see the same exponential growth of temperature as we did with weights.

Now, consider the circuit obtained by taking the graph over which we are trying to calculate maximum flow, connecting the source and the sink with a 1 volt voltage source, and turning each edge into a thermistor with $\alpha_e = 1/u_e^2$. This circuit will be in a steady state if $\frac{dT_e}{dt} = 0$ for each edge. Looking at equation (7), we solve:

$$\begin{aligned} \frac{dT_e}{dt} &= 0 \\ \left(\frac{f_e^2}{u_e^2} - 1\right) T_e &= 0 \\ T_e = 0 \text{ or } \frac{f_e}{u_e} &= 1 \end{aligned} \tag{8}$$

This is quite an interesting result. It means that in a steady state, each edge either has temperature 0 or flows exactly its capacity. In particular, any maximum flow corresponds to a steady state flow in which edge that is fully saturated has positive temperature and all other edges will have 0 temperature. In the graph shown below, we fix the voltage across s and t to be 1, so $v_s = 1$ and $v_t = 0$. If $T_{(a,t)} = 0$ then $r_{(a,t)} = 0$ and so $v_a = v_t = 0$. Now, if the capacity 2 edge flows 2 units of flow, then its resistance must be $r_e = V/I = 1/2$. So, its temperature must be $1/2 {*} u_e^2 = 1/2 {*} 4 = 2 = u_e$. Similarly, if the capacity 1 edge flows 1 unit of flow it will have $T_e = u_e = 1$ and $r_e = 1$. In this configuration, we will be in a steady state, with 3 units of flow moving from s to t (note, with 0 resistance, the capacity 5 edge absorbs the 3 units of flow in order to preserve conservation of flow).

Figure 6: Simple graph with possible unfeasible steady state



It is worth noticing that the results of the above analysis are especially clean. Not only does the unsaturated edge have resistance 0, but the saturated edges each have a full voltage drop of 1 across them and have $T_e = u_e$. In fact, given any graph and set of edge capacities, if we randomly perturb the capacities to any extent, we will obtain with probability 1 a graph whose thermistor model has a similarly simple steady state. In the chain graph shown in Figure 3, all edges are saturated in the maximum flow. So there is a steady state of the thermistor circuit in which each edge has positive temperature and some voltage drop across it. However, if we randomly perturb the edge capacities, each chain will, with probability 1, have a single bottleneck edge that is saturated at maximum flow. All other edges will be unsaturated. We will then have a simple steady state in which each path from s to t has a single edge with $T_e = u_e$ and $v_e = 1$. All other edges will have $T_e = 0$. We show that this simple steady state exists for general graphs.

Theorem 5.1. *Given any graph and associated set of edge capacities, a random perturbation of edge capacities will with probability* 1 *give us a new graph whose equivalent thermistor circuit has for any steady state configuration, for any edge, either $T_e = u_e$, $v_e = 1$, and $f_e = u_e$, or $T_e = v_e = 0$.*

Proof. With a random perturbation of edge capacities, the probability that the sum of any set of edge capacities exactly equals another capacity (or sum of other capacities) will be 0. Consider some steady state configuration for the circuit. Take every non-exactly saturated edge $e = (u, v)$ with $T_e = r_e = 0$, and remove it from the graph by combining u and v into one node. We allow multiple edges between nodes, so if u and v were both connected to a third vertex a, we leave two edges between a and the new combined node. This operation of collapsing out $T_e = 0$ edges will not change the voltage, and therefore the current, across any exactly saturated edge (any edge with $f_e = u_e$) since any two combined vertices must have had the same voltage in the original graph since they were connected by an edge with $r_e = 0$. Current flow between these nodes was entirely free, so from the view point of the rest of the circuit, the edges coming out of these nodes behaved exactly as multiple edges coming out of one node. Further, the collapsing operation will not remove any exactly saturated edges. This could only happen if the saturated edge connected two vertices with the same voltage in the original steady state configuration. However, without a voltage drop, any edge with positive resistance would not flow any current so could not be saturated. (Technically, if the exactly saturated edge had resistance 0 it could have flow across it. However, if we just assume that if two zero resistance edges flow between two nodes and at least one is not saturated, then they split the flow so that neither is saturated, then this case cannot exist.)

Now, after the collapsing operation is complete, we will be left with a graph containing only edges flowing at exactly full capacity. We argue that this collapsed graph will only contain two nodes - s and t, so all of these edges must flow directly from the source to sink. If the graph contained any node, v other than s and t, this node would have to obey the law of conservation of flow. It would have to have $\sum_{e:e=(v,u)} I_{in} u_e = 0$, where $I_{in} = 1$ if the edge is flowing current into v and $I_{in} = -1$ if the current flows out of v. However, our random perturbation insures that, with probability 1, this sum can never exactly equal 0. So, we cannot have any internal nodes.

As a consequence, we see that in the original graph all saturated edges with positive temperature must have had a voltage drop of 1. They have a voltage drop of 1 in the collapsed graph and the collapsing operations, as explained above, did not change the voltage across any edge. A voltage drop of 1 means that for any saturated edge, in the steady state we have:

$$\begin{aligned} v_e &= f_e * r_e \\ 1 &= u_e * \frac{1}{u_e^2} T_e \\ T_e &= u_e \end{aligned} \tag{9}$$

□

Unfortunately there can be many steady states of even a simple thermistor circuit like the one shown above. For example, if we simply set the temperatures on the capacity 1 and capacity 2 edges to 0, and the temperature of the capacity 5 edge to 5, then we will have $v_a = v_s = 1$. So the capacity 5 edge will flow $\frac{v_a - v_t}{r_e} = \frac{1}{1/5} = 5$ units of flow. This will clearly cause a capacity violation on at least one of the two edges leading into a.

However, these infeasible steady states are in a sense 'fragile'. If any edge is flowing over capacity in a steady state then we must have $T_e = 0$ since $\left(\frac{f_e^2}{u_e^2} - 1\right)$ must be positive. Now, any deviation from the 0 temperature will cause dT_e/dt to become positive. This will cause temperature to increase, further increasing dT_e/dt. In contrast, in a feasible steady state, any edge with $T_e = 0$ will be flowing under capacity. So if T_e increases slightly, dT_e/dt will be negative since $\left(\frac{f_e^2}{u_e^2} - 1\right)$ will be negative. This negative derivative will tend to pull the temperature back down to 0.

If we add back in a positive ambient temperature then we see that infeasible steady states are eliminated altogether.

Theorem 5.2. *If $T_a > 0$ then every steady state flow f has $f_e/u_e \leq 1$ for all $e \in E$*

Proof. To be in a steady state we need:

$$\begin{aligned} \left(\frac{f_e^2}{u_e^2} - 1\right) T_e + T_a &= 0 \\ \left(\frac{f_e^2}{u_e^2} - 1\right) T_e &= -T_a \end{aligned} \tag{10}$$

This cannot hold for any edge flowing over capacity since we will have $\left(\frac{f_e^2}{u_e^2} - 1\right) > 0$. Further, if we start with all positive edge temperatures (naturally, if we start with $T_e = T_a$ for all edges) then we must have $T_e \geq 0$ since

dT_e/dt goes to 0 as the temperature goes to 0. So, for an over capacity edge we will always have $\left(\frac{f_e^2}{u_e^2} - 1\right) T_e \geq 0 > -T_a$. So we can have no overcapacity edges in a steady state.

□

Further, if we set T_a very small, then it seems very likely that any steady state flow will be an approximate maximum flow. We have not yet proved this fact but we show the related fact:

Theorem 5.3. *If $T_a = 0$ then any feasible steady state flow f will be a maximum flow.*

Proof. As described above, with a slight random perturbation of our edge capacities, any feasible steady state will have all edges either saturated with $T_e = u_e$ or unsaturated with $T_e = 0$ (since we are feasible, we don't need to consider over-saturated edges with $T_e = 0$). Now, assume by way of contradiction that we have a feasible steady state flow f that is not a maximum flow. As is well known in the theory of maximum flow (and used for example as the basis for the Ford-Fulkerson algorithm), f must have some augmenting flow path from s to t. For a first case, assume that this path only contains edges that are unsaturated in f. Then, the resistance along the path is 0, which would lead to unbounded flow between s and t in f. So such an unsaturated path cannot exist. So any augmenting path must include at least one saturated edge e. The voltage drop across e in f is 1. In order to not violate capacity constraints, the augmenting path must flow across e from its 0 volt node u_0 towards its 1 volt node u_1. Now, let p_0 be section of the augmenting path leading from s to u_0. p_0 must include at least one saturated edge since s and u_0 have different voltages in f. However, the voltage drop across this second saturated edge must be in the direction of flow along the augmented path - from s to u_0. This edge is already saturated in this direction. So our augmenting path cannot be valid.

So, f cannot contain any augmenting paths. So, f must in fact be a maximum flow. □

Although it is yet to be proven, it seems likely that as we reduce T_a to a very small positive value, we will ensure that any steady state flow is feasible, but also that the steady state flows converge towards a maximum flow. Specifically, it seems that we should be able to show that f contains no augmenting paths above a certain capacity (which is a function of T_a). Using this, we should be able to lower bound the flow throughput of f and show that it is near maximal for small T_a.

5.1 Basic Forward Simulation Algorithm Using Thermistor Circuits

The above facts and conjecture suggest a very simple algorithm to compute maximum flow using thermistor circuits. If we simply start with our circuit in some initial state, for example with $T_e = T_a$ for all edges, or simply T_e very high for all edges, we can then step forward in small discrete steps, recalculating temperatures and flows at each step according to the relevant differential equations. Choosing a small T_a should cause our steady state flow to be an approximate maximum flow. Of course there are some difficulties with this method and facts that would need to be proven to show that the method actually converges on a maximum flow.

Proof that steady state is approximate maximum flow As explained above, this algorithm relies on the assumption that any steady state flow with $T_a > 0$ but very small will be an approximate maximum flow.

Convergence on steady state Even after knowing that every steady state is a maximum flow, to show that forward simulation works we would need to show that the circuit eventually enters the steady state configuration and does not oscillate out of steady state. It seems intuitive that this would be true, and it may even be know in the electronics literature, however we have yet to prove or find a proof that a pure thermistor circuit with a single voltage source always reaches a steady state configuration.

Step size for forward simulation It would be necessary to find a step size (or a function giving step size at any given iteration) that allows for accurate simulation of the circuit so that approximate convergence on steady state occurs. Determining the step size would also be the key determinant of the algorithm's runtime.

Zero-resistance edges Edges with zero resistance become a problem numerically because as r_e goes to 0, the conductance $c_e = 1/r_e$ goes to infinity - so the elements in the Laplacian matrix go to infinity. One way around this problem is to collapse two nodes into a single node once the resistance between them is 0. Once the resistance is 0 it can never change as we will have $T_e = 0$ so dT_e/dt will be fixed at 0. Of course, there would need to be a reasonable way to decide when the resistance between two nodes is small enough that we can truncate to 0 and combine the nodes. Another option (which I used in my test implementation of the thermistor flow algorithm) is to simple choose a very low lower bound for resistance. Perhaps it is possible to prove that approximating zero resistance by a very low value will not have a large effect on the steady state flow.

As a proof of concept I did implement a very simple algorithm that uses forward simulation to calculate maximum flow from a thermistor circuit. Figure 7 below shows a very basic graph on which I tested the algorithm. At each step, we recalculate the electric flow given current temperatures and then for each edge set:

$$T_e(i+1) = T_e(i) + \epsilon \left(\frac{f_e^2}{u_e^2} - 1 \right) T_e(i)$$

where $\epsilon = 10^{-3}$ and $T_e(0) = 1$ for all edges. (Note: here $T_a = 0$). To prevent a blow up in conductance values, we set $r_e = \min(r_e, 10^{-6})$ at each step. The maximum flow on the graph is 11, with 6 flowing along $(1,2)$, 5 flowing along $(1,3)$, 1 flowing along $(2,3)$ (towards 3), and 5 and 6 flowing along $(2,4)$ and $(3,4)$ respectively. The simulated thermistor circuit correctly converges on this final flow. In Figure 8 we show the convergence of flow values (noting that due to symmetry the $u = 10$ edges and the $u = 5$ edges have the same flow at each step of the algorithm so there trajectories overlap. In Figure 9 we see that the unsaturated $u = 10$ edges converge to $T_e = 0$. The saturated $u = 5$ and $u = 1$ edges converge to $T_e = u_e$.

Figure 7: A simple graph on which to test the thermistor flow algorithm

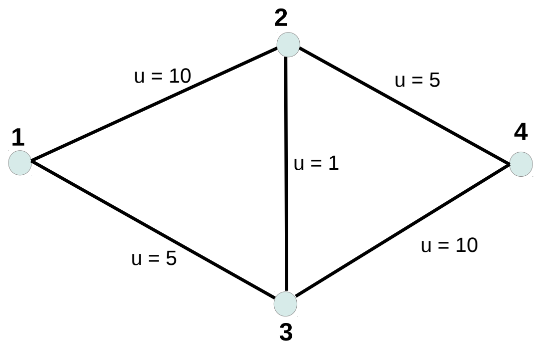


Figure 8: Convergence of flow values towards thermistor circuit steady state

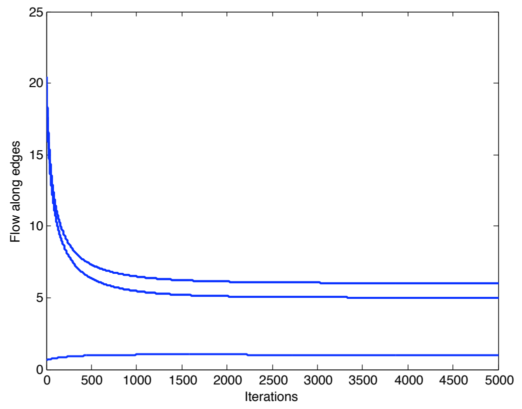
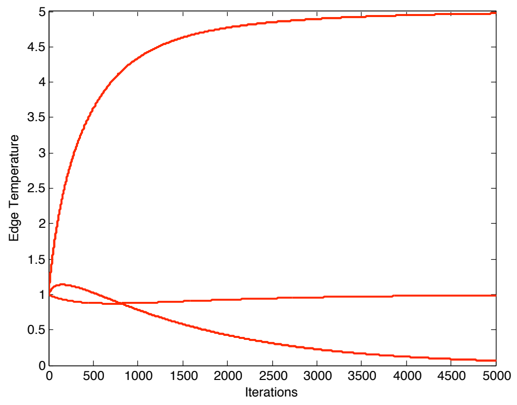


Figure 9: Convergence of temperature values towards steady state



5.2 Other Possible Thermistor Algorithms

Of course, despite the simplicity of forward simulation, there may be other much more effective ways of finding the steady state of our thermistor circuit. One possibility would be to use a sort of 'dual update' algorithm in which we start with a high ambient temperature and decrease it towards 0 over the course of the algorithm. The initial high ambient temperature would possibly enforce that $\left(\frac{f_e^2}{u_e^2} - 1\right)$ is kept well below 0. As ambient temperature was decreased, it would allow the flow on the circuit to approach saturation on some edges. This seems very analogous to the decrementing of the weight of a log barrier function used in potential reduction interior point algorithms such as described in [1].

6 Conclusion and Future Work

Obviously, nearly all the work presented in this report is incomplete and could be further developed. A proof that we can avoid binary search for the faster $\tilde{O}(m^{4/3}\epsilon^{-11/3})$ edge cutting algorithm would be interesting. In addition, even if it does not improve the runtime of the algorithm, a proof (or counter example disproving) that the electrical flows eventually converge to an approximate maximum flow and settle down to a steady state rather than oscillate would be key to understanding the algorithm's behavior. It seems very intuitive that the electrical flows should eventually converge to a steady state - even if this is not within the number of iterations described in the algorithm. Perhaps looking at the continuous time process analogous to the iterative algorithm will help reveal more about that answer to this question.

The connection between maximum flow and the steady states of thermistor networks seems to be an interesting area to explore. Perhaps that direction will lead to an alternative maximum flow algorithm using the slightly modified difference equations but more closely tied to a real world physical phenomena. The open questions raised in the thermistor section seem very approachable. Exploring alternative methods for solving thermistor steady states could bring new ideas to solving the maximum flow problem.

Finally, there is much work to be done in understanding the connection between this specific maximum flow algorithm and general interior point algorithms. This is in general a difficult area to explore as useful empirical tests can be difficult to design and run. Further, the algorithms can be framed in such different lights and proved correct using such different techniques that it can be difficult to clearly see the connections between their operation. However, there could be a lot to learn about both the interior point algorithms and about this algorithm by relating them to each other.

7 Code Appendix

This appendix includes a short description of the pieces of code that I produced for this project, just for the purpose of cataloging the work.

7.1 Flow Algorithms

`approx_flow.m` Implements the original flow approximation algorithm described in [2]. If `cut` is set to 1 then uses the faster $\tilde{O}(m^{4/3}\epsilon^{-11/3})$ algorithm. Otherwise uses the more basic algorithm. Does not perform graph smoothing to get best runtime.

`approx_cut.m` Implements the dual approximate minimum cut algorithm as described in [2]

`check_flow_no_fail.m` Implements the original flow approximation algorithm without binary search (and without edge cutting). Maintains a flow goal that is scaled down when necessary and used as the value for computed electric flows.

`check_flow_no_fail_upscale.m` Implements the original algorithm without binary search exactly as described in this paper. Scales each electrical flow to have energy exactly $(1+\epsilon)|\boldsymbol{w}|_1$

`linear_prog_flow.m` Calculates the maximum flow using `Matlab`'s built in linear programming function.

`thermistor_flow.m` Very basic forward simulation of a thermistor network to compute maximum flow. Approximates 0 resistance values with very small resistances, rather than with edge collapse.

7.2 Flow Utilities

`check_flow.m` Called by the `approx_flow.m` algorithm to find a feasible flow of value F. This value is given in a flow vector `x` which has value F in position s and $-F$ in position t. Fails if $F > F^*$

`crude_flowbound.m` Finds a crude flow lower bound (0) and upper bound (minimum of the sum of capacities into s and out of t). Used by `approx_flow.m` to bound binary search.

`electric_flow.m` Calculates an electric flow over a set of resistances. Uses the standard `Matlab` backslash solver unless `cong_flag` is set to 1, in which case it uses a preconditioned conjugate gradient solver with an incomplete Cholesky preconditioner.

`sqrtm_oracle.m` Uses edge weights to calculate resistances as prescribed in [2]. Returns the electric flow over these resistances. Performs edge cutting if `cut` flag is set. Returns $f = 0$ if the calculated flow has energy above the flow bound.

`sqrtm_oracle_no_fail.m` Same as `sqrtm_oracle.m` except never fails. Simply returns a flag if the electric flow is above the energy bound.

7.3 Interior Point Algorithms

`affine_pra.m` Solves maximum flow on a *directed* graph using the Affine Potential Reduction Algorithm described in [1]. Uses `Matlab` backslash solver for linear solves unless `cong_flag` is set, in which case it uses a preconditioned conjugate gradient method.

`affine_pra_undir.m` Same as `affine_pra.m` except works on an undirected graph. Uses the linear programming formulation described in this paper, which produces a smaller program than simply converting to a directed graph and running `affine_pra.m`.

7.4 Other Utilities

`a2u.m` Converts an $n \times n$ graph adjacency matrix to an $n \times m$ vertex edge transfer matrix (B as used in this paper).

`count_osc.m` Basic utility code that counts how many times the flow values on a set of edges cross their means. Used to find edges that may possibly be experiencing oscillating flow.

`display_graph.m` Plots a graph's adjacency matrix using `spy`. Draws a graph in 2 dimensions using all possible combinations of the first 20 Laplacian eigenvectors.

`udir2dir.m` Converts vertex edge transfer matrix and capacity vector for undirected graph into the appropriate values for the corresponding directed graph (with each edge replaced by two edges flowing in opposite directions).

`weight_plot.m` Plots a graph over a given set of vertex coordinates. Color codes edges bases on weights.

`combine_graphs.m` Takes in vertex edge transfer matrices and capacity vectors for two graphs. Joins the graphs by mapping a specified vertex from one onto a specified vertex of the other.

`edge_to_graph.m` Takes in vertex edge transfer matrices and capacity vectors for two graphs. Replaces a specified edge from one graph with a full copy of the second graph.

`netflow2params.m` Converts a graph from the form produced by the Netgen program [7] to a vertex edge transfer matrix and capacity vector.

`bad_chains.m` Produces a graph with a specified number of edge chains each with a specified length running directly from s to t.

`bad_kgraph.m` Builds a graph as shown in Figure 3 of [2]. Graph has k edge chains of length k along with a single chain of length 1 all leading from source to sink.

References

[1] Anstreicher, Kurt M. Potential Reduction Algorithms. Chapter 4 in *Interior Point Methods of Mathematical Programming*, T. Terlaky, ed., Kluwer. 1996.

[2] Christiano, Kelner, Madry, Spielman and Teng. Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs. Proceedings of the 43rd annual ACM Symposium on Theory of Computing. (2010).

[3] Daitch, Samuel I. and Spielman, Daniel A. : Faster lossy generalized flow via interior point algorithms. STOC 2008: 451-460.

[4] D.R. Karger. Better random sampling algorithms for flows in undirected graphs. In *Proceedings of the 9th Annual Symposium on Discrete Algotihms*, pages 490-499. Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[5] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving SDD systems. In Proceedings of the 51st Annual Symposium on Foundations of Computer Science, 2010.

[6] Klingman, Napier, and Stutz. Netgen: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems. Management Science. Vol. 20, No. 5, January 1974.

[7] Network Flows and Matching. The First DIMACS Implementation Challenge. 1990-1991. ftp://dimacs.rutgers.edu/pub/netflow. (Includes maximum flow and minimum cost flow code, test cases, and graph generators.)

[8] Reenstra, Arthur L. A Low-Frequency Oscillator Using PTC and NTC Thermistors. *IEEE Transactions on Electron Devices*, Vol. ED-16, No. 6, Jine 1969.

[9] Renegar, James. A Polynomial-Time Algorithm, Based on Newton's Method, for Linear Programming. *Mathematical Programming* Volume 40 Issue 1, January 1988.

[10] Thermistor. http://en.wikipedia.org/wiki/Thermistor. Accessed April, 2012.

[11] Tzallas-Regas, George. Primal-Dual Interior Point algorithms for Linear Programming. www.doc.ic.ac.uk/~br/berc/pdiplin.pdf.

[12] Wayne, Kevin. Theory of Algorithms Lecture Slides. Spring 2011. http://www.cs.princeton.edu/~wayne/cs423/lectures/max-flow-applications-4up.pdf